Crowdsourcing Complex Workflows under Budget Constraints

Long Tran-Thanh
University of Southampton, UK
ltt08r@ecs.soton.ac.uk

Trung Dong Huynh
University of Southampton, UK
tdh@ecs.soton.ac.uk

Avi Rosenfeld
Jerusalem College of Technology, Israel
rosenfa@jct.ac.il

Sarvapali D. Ramchurn
University of Southampton, UK
sdr@ecs.soton.ac.uk

Nicholas R. Jennings
University of Southampton, UK
nrj@ecs.soton.ac.uk

Abstract

We consider the problem of task allocation in crowdsourcing systems with multiple complex workflows, each of which consists of a set of inter-dependent micro-tasks. We propose Budgeteer, an algorithm to solve this problem under a budget constraint. In particular, our algorithm first calculates an efficient way to allocate budget to each workflow. It then determines the number of inter-dependent micro-tasks and the price to pay for each task within each workflow, given the corresponding budget constraints. We empirically evaluate it on a well-known crowdsourcing-based text correction workflow using Amazon Mechanical Turk, and show that Budgeteer can achieve similar levels of accuracy to current benchmarks, but is on average 45% cheaper.

Introduction

Major organisations are increasingly reliant on an online workforce to carry out computationally hard tasks (e.g., transcribing text, writing reviews, designing logos, or tagging images). Using online *crowdsourcing* platforms, these organisations are able to leverage millions of workers from around the world to carry out small units of work called micro-tasks in return for a small monetary reward (Bernstein et al. 2010; Lin, Mausam, and Weld 2012; Little et al. 2009; Ramchurn et al. 2013). By so doing, they are able to scale their workforce at will and minimise their costs.

When allocating tasks to a large online 'crowd', however, organisations face a number of challenges (Ho and Wortman-Vaughan 2012; Tran-Thanh et al. 2012; 2013). First, given that workers may come from different countries (e.g., China, India, UK, US) and have different skills, the quality of their work may vary significantly. For example, native English speakers are likely to perform better at correcting spelling mistakes or graduate graphic designers may do a better job at drawing a logo compared to a non-expert. Second, in many cases, it is not clear what determines whether the task is completed correctly or not (i.e., it is either too costly to determine or there is no ground truth), particularly when workers may perform the same task differently. Third, workers at differenet economic background may perform tasks to different quality levels depending on the rewards offered. Fourth, tasks may not be easily broken down into micro-tasks that can be independently completed by individual workers. This requires *multiple* complex *workflows* to be interleaved to complete the *interdependent* micro-tasks (e.g., the correction of spelling mistakes in a piece of text can be broken down into multiple workflows involving finding and fixing the mistakes or text can be summarised by deleting words or sentences, and writing new ones (Bernstein et al. 2010)). In general, these challenges point to the fact that crowdsourcing, even in the case of the simplest tasks, is error-prone and potentially costly.

While Kamar, Hacker, and Horvitz (2012), Dai et al. (2013), Karger, Oh, and Shah (2011), Simpson et al. (2011) propose using probabilistic techniques and redundancy (i.e., different workers are asked to perform the same task until consensus is achieved on the outcome) to deal with inaccuracies and minimise the costs incurred, they assume that individual tasks are independent from each other and can be arbitrarily allocated to the crowd (see next section for more details). Hence these solutions cannot be readily applied to problems involving inter-dependent micro-tasks. Instead, Bernstein et al.; Lin, Mausam, and Weld; Ramchurn et al. propose workflows to allocate interdependent micro-tasks in multiple phases, where each task performance phase generates new micro-tasks for the next phase. For example, in Bernstein et al. (2010) (and later extended by (Little et al. 2009; Lin, Mausam, and Weld 2012; Ramchurn et al. 2013)), a Find-Fix-Verify (FFV) workflow is used to correct and shorten text by passing 'Finds' of mistakes in sentences by a set of workers to another set of workers who 'Fix' these mistakes. In addition, another set of workers 'Verify' these mistakes (more details are given in the next section). In such a workflow, however, given that new tasks are created in some phases by the crowd, the cost of the exercise may grow beyond control.[1] To date, most approaches set limits on the number of tasks to be completed in each phase in an ad hoc fashion, and by so doing, trade off any guarantees of accuracy. In contrast, Tran-Thanh et al. proposed the BudgetFix mechanism to allocate a budget across multiple phases of a workflow and this was shown to cost less than and generate better results than (Bernstein et al. 2010). However, BudgetFix assumes prior knowledge of the difficulty of each task, sets arbitrary parameters to constrain the number of tasks for each phase of a workflow, and only considers how the budget is split across each phase of one workflow at a time. Thus,

[1] In the FFV workflow, for each Find, there are multiple possible Fixes and each Fix requires a number of possible verifications.

in general, previous approaches are parameterised, necessitating manual tuning, and do not attempt to manage costs in a principled way across multiple workflows.

Against this background, this paper addresses the problem of multi-workflow task allocation under budget constraints. In particular, we aim to solve a text correction problem previously studied by (Bernstein et al. 2010; Tran-Thanh et al. 2014)[2]. However, while the latter only consider one sentence at a time, we consider the more natural setting where text, consisting of *multiple* sentences, needs to be corrected using a FFV workflow for each sentence. Hence the challenge we address is to determine how a budget should be split *across* sentence-correction workflows, and *within* each workflow (i.e., for each Find, Fix, and Verify phase) in order to generate high quality outcomes. To this end, we propose Budgeteer, a budget and task allocation algorithm that uses an initial exploration phase to estimate the difficulty of the sentences. Based on these findings, it then chooses an allocation scheme that minimises a certain upper bound of the total error probability, i.e., the probability that we will provide an erroneous correction (for more details, see the next sections). Once the budgets are allocated to the sentences, Budgeteer allocates the number of Find, Fix, and Verify tasks for each of the sentences in a similar vein to BudgetFix. Note that our algorithm can efficiently (i.e., with negligible cost) learn the size of budget it needs to allocate to each of the sentences, and thus, it indeed can tune the parameters to adapt to different sentences.

More specifically, this paper advances the state of the art in the following ways. First, Budgeteer is the first budget-limited crowdsourcing algorithm to consider budget allocation across and within multiple complex workflows. Second, compared to existing benchmarks, Budgeteer does not rely on manually-tuned parameters to allocate the budget. Finally, through empirical evaluation using Amazon Mechanical Turk, we show that it achieves similar accuracy to the current benchmarks, but with 45% less cost.

Background

Here we first discuss some examples of complex crowdsourcing workflows and then present budget-constrained crowdsourcing algorithms. By so doing, we describe the problem addressed by Budgeteer and key benchmarks against which we evaluate it.

Complex Crowdsourcing Workflows

The most basic crowdsourcing workflows involve only one type of micro-task, for example, to tag an image or translate a sentence. However, as organisations attempt to address more complex problems, more complex workflows are required to recruit workers to solve these problems. For example, a number of techniques have been proposed to organise a workforce to follow certain constraints when planning itineraries (Zhang et al. 2012), to retain a crowd and select a pilot to drive a robot in real-time (Lasecki et al. 2012), and to correct and summarise text within a text editor (Bernstein et al. 2010). We use the latter domain as it is a well-established benchmark for crowdsourcing. In particular, we focus on the text correction element of the Soylent system designed by (Bernstein et al. 2010) and later adopted by (Tran-Thanh et al. 2014; Ramchurn et al. 2013). Soylent uses the FFV workflow to break a large, complex task into Find, Fix and Verify micro-tasks (see previous section for more details). To address the possibility that multiple phases will generate exponentially increasing numbers of crowdsourcing tasks, hence resulting in uncontrollable costs, Soylent relies on very simple ad hoc heuristics (e.g., only 'find's with 20% agreement are passed to the Fix phase) that determine the number of outcomes to pass from one phase to another. While this helps achieve high accuracy of the outcomes in practice, it is by no means budget-efficient. In contrast, our approach tackles same problem of task allocation but with a budget constraint and therefore determines how this budget should be split using FFV to maximise accuracy.

Budget-Constrained Crowdsourcing

A number of approaches have emerged in the last few years to consider crowdsourcing under budget constraints (Karger, Oh, and Shah 2011; Ho and Wortman-Vaughan 2012; Tran-Thanh et al. 2012; 2013; Azaria, Aumann, and Kraus 2014). While most of these approaches attempt to provide some performance guarantees (i.e., in terms of task completion accuracy), they typically apply to simple crowdsourcing workflows whereby workers are only asked to do one type of task rather than the heterogenous micro-tasks that are interleaved in complex workflows. Moreover, they assume that all budgeted tasks have an equal level of difficulty. More recently, the BudgetFix algorithm (Tran-Thanh et al. 2014) addressed the problem of budget-constrained crowdsourcing for complex workflows. While making no assumptions about task difficulty, they rely heavily on finely-tuned parameters to guarantee high quality results. In fact, as we show in our evaluation, if such parameters are poorly selected, BudgetFix can perform worse than ad hoc methods used by other approaches in the literature. Moreover, BudgetFix only considers how to allocate a given budget within one workflow. In contrast, Budgeteer automatically adapts to the difficulty of tasks and allocates the budget *across* multiple workflows.

Problem Definition

We now formally define the problem of task allocation with multiple workflows under budget constraints for the text correction problem presented in the previous section. Furthermore, for the sake of simplicity, we assume that there is only one mistake to fix within a given sentence.

Note that it is not difficult to relax this assumption to the case of multiple mistakes per sentence. In fact, we can replace the one-winner selection tournament methods we use in our algorithm (see the subsequent section for more details) with multi-winner selection techniques, such as given in (Kaufmann and Kalyanakrishnan 2013; Kalyanakrishnan et al. 2012). In what follows, we first present the FFV workflow and then describe the constraints and assumptions under which it is executed.

The Find-Fix-Verify Workflow

For now we only consider one sentence (and later consider multiple sentences when having to distribute the budget). Our goal here is to find a single mistake within this sentence. The FFV workflow allows a task allocation agent to

[2]While we focus on a text correction problem, our approach generalises to other tasks requiring complex workflows.

separate the tasks of finding a mistake, correcting it, and ensuring that all corrections to fixes received are themselves correct. This workflow was shown to be effective in getting text corrected to a high accuracy (over 90%) without any budget limitations. Here we formalise and then extend it to consider budget constraints. The details of each step are:

Find: the agent asks the workers to identify the position of the error in sentence s. Let X denote the set of possible error candidates (i.e., all positions in the text at which an error can occur). We assume that these responses are drawn from an *unknown* distribution D^X, and that the error is at true position x^t.

Fix: the agent asks the workers to fix the identified errors at positions $X' \subseteq X$ (we may not request all the errors found to be fixed). In addition, for each $x \in X'$ possible error position, let $Y(x)$ denote the set of possible fixes that belong to position x. We assume that for each $x \in X'$, $Y(x)$ contains the "No-Fix" response. This response represents the case when x is believed to be non-erroneous. We assume that for each $x \in X'$, the fix responses that belong to x are also drawn from an unknown distribution $D^{Y(x)}$. For each $x \in X'$ we denote its true fix $y^t(x)$.

Verify: the agent asks the workers to vote on whether a possible error-fix pair (x, y) is correct. Let $Z(x, y)$ denote the set of verification responses of pair (x, y). For each $z \in Z(x, y)$, we have $z = 1$ if the voter thinks the fix y of position x is correct, i.e. $y(x) = y^t(x)$, and $z = 0$ otherwise. Thus, the responses that belong to each pair (x, y) can be regarded as random variables drawn from an unknown Bernoulli distribution $D^{Z(x,y)}$.

The FFV workflow only applies to one sentence at a time and hence, previous approaches have only considered how to allocate a budget to each step of the workflow (Bernstein et al. 2010; Tran-Thanh et al. 2014). This ignores the fact that different sentences have different difficulties and an organisation will dedicate a budget to a collection of sentences rather than to individual sentences at a time. Hence, in the next section, we formalise the problem of crowdsourcing tasks under budget constraints for multiple workflows.

Multiple Workflows under a Budget Constraint

We now turn to the description of our problem. Let the costs for requesting a task in each of the Find, Fix, and Verify phases be defined as real values c^X, c^Y, and c^Z, respectively (these costs are the same for all the sentences within the text). Suppose that the text that needs to be corrected consists of S^0 sentences, from the set $\mathbb{S}^0 = \{1, \ldots, S^0\}$. For each $s \in \mathbb{S}^0$, let N_s^X, N_s^Y, and N_s^Z denote the number of responses we require from the crowd for each of the FFV phases for sentence s, respectively.

Now, given a budget $B^0 \in \Re^+$, the total cost of the requested tasks should not exceed this budget. This can be expressed as the following constraint:

$$\sum_{s=1}^{S^0} \left(N_s^X c^X + N_s^Y c^Y + N_s^Z c^Z \right) \leq B^0 \qquad (1)$$

The challenge is then to determine the values of N_s^X, N_s^Y, and N_s^Z for each $s \in \mathbb{S}$ so that this constraint is not violated. Crucially, it is important to ensure that each sentence gets enough coverage by the crowd in order to find and fix any mistakes in it. Hence, we describe the Budgeteer algorithm that solves this problem.

The Budgeteer Algorithm

Budgeteer first calculates how to efficiently split the budget among the sentences such that each budget is sufficiently large for good error detection within the corresponding sentence. It then uses the assigned budget to allocate Find, Fix, and Verify tasks for each of the sentences by solving an optimisation problem. The main challenge here is to find a good way to split the budget across multiple runs of FFV on individual sentences. In particular, as observed by Tran-Thanh et al. (2014), difficult sentences (those that involve more complex words or grammar) tend to require more tasks to be completed. This is due to the fact that the crowd contains workers that are not native English speakers that may not be able to differentiate between mistakes and complex words. Hence, the budget split across multiple sentences needs to capture the difficulty levels of individual sentences in order to ensure each sentence gets enough passes through the FFV workflow for accurate error correction. Given this, Budgeteer consists of: (i) an exploration phase, where we use a small portion of the budget to estimate the difficulty of the sentences; (ii) a budget allocation phase, where Budgeteer calculates how much budget it should assign to each sentence; and (iii) an optimisation phase, where the algorithm solves an optimisation problem to derive the number of Fix, Find and Verify tasks for each particular sentence.

1. Exploration phase: We use a uniform exploration approach by requesting a fixed, small number of Finds per sentence. Note that in our experiments, we use three Finds/sentence (see the next section for more detail), and for those sentences with one Find candidate (i.e., all the responses suggest the same location as candidate), we request the same number of Fix tasks per sentence. We repeat this with sentences with one Fix candidate in their Verify phase.

The intuition behind this exploration phase is that, by crowdsourcing a small number of micro-tasks for each sentence, we can build an initial estimate about the difficulty of the sentences. In particular, those with a larger number of Find or Fix candidates can be considered as more difficult sentences, and vice versa (see, e.g., Figure 2 for more details). On the other hand, we can easily spot the extremely easy sentences, i.e., those with 1-1 Find and Fix candidates. The key challenge here is to ensure that this exploration phase does not significantly increase the cost of the whole exercise. However, in what follows we argue that we can already perform error correction with high accuracy from the responses collected within this phase, and thus, we do not need to allocate additional budget to these sentences within the next phases.

2. Budget allocation phase: We consider $\mathbb{S}$, the set of sentences that still require additional budget allocation after the exploration phase ($\mathbb{S} \subseteq \mathbb{S}^0$). Note that after the exploration phase, we do not request any additional tasks for the extremely easy sentences. For each sentence $s \in \mathbb{S}$, let $K_{\max}^s$ and $L_{\max}^s$ denote the number of Find and Fix candidates of sentence s we received in the exploration phase. If we have

at least one Find candidate for s in the exploration phase, we can set $K^s_{\max}$ as the number of Find candidates of s. Otherwise, we set $K^s_{\max} = \max_{s'} K^{s'}_{\max}$, where s' indexes through all the sentences with a positive number of Find candidates. We can set $L^s_{\max}$ in a similar way (but with Fix candidates). By so doing, we define $K^s_{\max}$ and $L^s_{\max}$ for each s. Given that the difficulty of a sentence is strongly related to the number of its Find/Fix candidates, $K^s_{\max}$ and $L^s_{\max}$ can, in fact, provide a good estimate of the difficulty of s. We now set the budget B^s for sentence s as follows. Let $B \leq B^0$ denote the total residual budget after the exploration phase. For each sentence $s \in \mathbb{S}$, let B^s denote the budget allocated to s. We have:

$$\begin{aligned} B^s &= C^s_1 \frac{B - \sum_{r\in\mathbb{S}} C^r_1 \left(\ln C^r_1 - \frac{C^r_1}{C^r_2} - \ln 3\right)}{\sum_{r\in\mathbb{S}} C^r_1} \\ &+ C^s_1 \left(\ln C^s_1 - \frac{C^s_1}{C^s_2} - \ln 3\right) \end{aligned} \quad (2)$$

where for each $s \in \mathbb{S}$, C^s_1 and C^s_2 can be calculated as follows:

$$C^s_1 = \sum_j \frac{c^j}{W^{s,j}}, \quad C^s_2 = \sum_j c^j \frac{V^{s,j} + \ln \frac{W^{s,j}}{c^j}}{W^{s,j}}$$

where j is either X, Y, or Z, and for each $s \in S$, we have:

$$\begin{aligned} W^{s,X} &= \frac{1}{2},\ V^{s,X} = \ln 2 \\ W^{s,Y} &= \frac{1}{v(K^s_{\max})^2},\ V^{s,Y} = \frac{1}{vK^s_{\max}} + \ln \frac{K^s_{\max}(K^s_{\max}-1)}{2} \\ W^{s,Z} &= \frac{1}{v(L^s_{\max})^2},\ V^{s,Z} = \frac{1}{vL^s_{\max}} + \ln \frac{L^s_{\max}(L^s_{\max}-1)}{2} \end{aligned}$$

The intuition of this budget allocation given in Equation (2) is as follows. Suppose that the budget for or each sentence s is given. In this case, we only need to focus on the task allocation of Find, Fix and Verify tasks within the workflow of each s, such that the total cost of the task allocation does not exceed the given budget. To to so, we solve an optimisation problem (which we describe later) in order to determine the efficient number of Find, Fix, and Verify tasks that we need to request from the crowd. In particular, the solution of this optimisation problem provides a task allocation that minimises an upper bound on the error probability of s, i.e., the probability that we incorrectly estimate either the location or the correction within s (for more detail, see Tran-Thanh et al. (2014)). Let us denote this bound with $P(B^s)$, where B^s is the budget allocated to s (see Equation (4)). Given this, the total error probability of $\mathbb{S}$ is upper bounded by

$$\sum_{s\in\mathbb{S}} P(B^s), \text{w.r.t.} \sum_{s\in\mathbb{S}} B^s \leq B \quad (3)$$

Thus, our budget allocation problem can be formalised as another optimisation problem, where the goal is to find a budget allocation such that the aforementioned total error bound defined in Equation (3) is minimised. This, in fact, can be done by using the standard Lagrangian relaxation method (Boyd and Vandenberghe 2004) which we do not detail here due to limited space. Thus, by solving this optimisation problem, we obtain that Equation (2) provides the optimal solution, and thus, the minimal bound of the total error probability bound.

It is worth to note that the budget allocated to a sentence s is proportional to its difficulty level as follows: Let the tuple $\langle k, l\rangle$ be the value of $K^s_{\max}$, and $L^s_{\max}$, respectively. In this case, the budget allocated to s is $B^s = O(k^2 \ln k + l^2 \ln l)$.

3. Optimisation phase: Having defined the budget B^s for each $s \in S$, we turn to the allocation of Find, Fix, and Verify tasks per sentence, with respect to its budget B^s. To do so, we use a state-of-the-art single-winner selection technique (Audibert, Bubeck, and Munos 2010) to choose the best candidate for Find, and then for Fix, respectively.

3.1 Estimation of Find: In particular, we first request Find tasks for s up to a limit N^X_s, which will be determined later. Let K_s denote the size of the set of Find candidates X_s (i.e., $K_s = |X_s|$). We then choose the best $\min\{K^s_{\max}, K_s\}$ Find candidates from X_s (the min operator is required as we do not want to exceed $K^s_{\max}$). This represents the set of possible candidate from which we would like to choose our estimate. We then run the following single-winner tournament to estimate the correct Find: We run the tournament in rounds such that we always eliminate one candidate at the end of each round. Within each round, we request the same, fixed number of Fix tasks for each of the remaining Find candidates such that the total number of Fix tasks requested within the whole tournament should not exceed N^Y_s, which will be determined later as well.[3] To eliminate the weak candidates at the end of each round, we measure the fitness of each Find candidate $x \in X_s$ by calculating the ratio of its total fixes that are not "No-Fix" (i.e., $y(x) \neq$ "No-Fix" — recall that a "No-Fix" is the identification that a sentence has no errors) to its total Fix responses (i.e., this fitness measures the probability that the location does indeed require a fix). The weakest candidate, and thus the one we eliminate, is the one that has the lowest fitness at the end of the round. When only one Find candidate remains, we choose it as our estimate.

3.2 Estimation of Fix: To estimate the correct fix, we run another single-winner selection tournament among the set of Fix candidates $Y_s(x^*)$ of the winning Find candidate x^*. In particular, let L_s denote the total number of these Fix candidates (i.e., $L_s = |Y_s(x^*)|$). Similar to the tournament above, we only run this tournament on the best $\min\{L^s_{\max}, L_s\}$ Fix candidates. The nature of the tournament is the same as the previous one, except for the fact that here we request Verify tasks to measure the fitness of the candidates, such that the total number of Verify tasks cannot exceed N^Z_s (also determined later), and the fitness is defined as the ratio of the candidate's positive responses (i.e., $z(y, x^*) = 1$) to its total number of Verifies. Again, similar to the previous tournament, the ratio of positive Verifies, and thus the fitness level, measures how likely it is that a particular Fix is correct. We also eliminate the weakest candidate at the end of each round. The last remaining Fix y^* will be our estimate.

3.3 Identifying the number of tasks per phase: We now turn to determine N^X_s,N^Y_s and N^Z_s, the number of Find, Fix, and Verify requests for s. Note that this step in fact has to be

[3] This fixed number of Fix tasks per Find candidate per round can be explicitly calculated from N^Y_s and $\min\{K^s_{\max}, K_s\}$.

Category:	Low	Medium	High	V.High
Flesch Reading Ease	90.4	56.8	36.8	0.53
Fog Scale Level	6.4	14.1	18.8	34.4
Flesch-Kincaid	4.2	10.2	14.9	30.7

Table 1: Complexity measures for sentences per category.

done before steps 3.1 and 3.2. In particular, suppose that we have done this step, then for any fixed tuple of N_s^X,N_s^Y,and N_s^Z, after running steps 3.1 and 3.2, by using the theoretical results from (Audibert, Bubeck, and Munos 2010), we can calculate an upper bound on the probability of not choosing the correct Find or Fix candidate (i.e., error probability). Let $P(N_s^X, N_s^Y, N_s^Z)$ denote this upper bound of error probability. Following (Tran-Thanh et al. 2014), we can identify the optimal value of N_s^X, N_s^Y and N_s^Z, such that $P(N_s^X, N_s^Y, N_s^Z)$ is minimised. In fact, we can obtain this optimal solution by solving the convex optimisation problem below (also by using the Lagrangian relaxation approach):

$$\min_{N_s^X, N_s^Y, N_s^Z} P(N_s^X, N_s^Y, N_s^Z), \text{ s.t. } N_s^X + N_s^Y + N_s^Z \leq B^s \quad (4)$$

Let $P(B^s)$ denote this minimal error probability bound. Recall that the value of $P(B^s)$ is used to solve another optimisation problem, whose solution leads to our budget allocation scheme given in Equation (2). Note that during this optimisation phase, we also take into account our observations from the exploration phase (i.e., we do not ignore any Find, Fix or Verify responses from the first phase).

Evaluation Settings: In order to evaluate BudgetFix within a realistic setting, we replicate an experiment similar to the text correction task previously used to evaluate the Soylent and BudgetFix algorithms. To do so, we create a dataset of a total of 100 sentences.[4] We inject mistakes in a similar manner to (Tran-Thanh et al. 2014). In more detail, we consider three types of mistakes with equal distribution: spelling mistakes based on an added letter, spelling mistakes based on a missing letter, and grammar mistakes. We place these mistakes into sentences that are of four different complexity levels which generally represent sentences from low to extremely high complexity (low, medium, high, very high). The low complexity sentences are taken from publicly available children's stories,[5] the medium ones are taken from open computer science textbooks,[6] the high ones are taken from recently published computer science articles, and the extremely high ones from the terms of conditions of e-commerce websites. To quantify the complexity of these sentences we measure the Flesch Reading Ease, Fog Scale Level, and Flesch-Kincaid Grade Level measures (Flesch 1948; Kincaid et al. 1975), which constitute accepted measures of sentence complexity (see Table 1).

Performance Comparison: We then apply the Soylent, BudgetFix, and Budgeteer algorithms to these sentences. As per the original Soylent experiments, we use Amazon Mechanical Turk (AMT 2010) and workers are paid the same amount, i.e. \$0.06 per Find task, \$0.08 for Fix tasks, and \$0.04 for Verify tasks. Within Soylent, regardless of the sentence difficulty or budget, a minimum of 10 Find, 5 Fix, and 5 Verify tasks are generated per sentence (as per (Bernstein et al. 2010)). In contrast, both BudgetFix and Budgeteer use variable numbers of Finds, Fixes and Verifies as per their algorithms. In order to minimise the element of chance, we requested more responses from the crowd than required by the three algorithms and simulate the algorithms 100 times, each time on a randomly picked set of responses from the pool of responses we received. The same method was also used to produce all the results later reported here.

[4]This dataset is available at `http://bit.ly/1sTya7F`.

[5]`http://bit.ly/1m4wEz2`.

[6]`http://bit.ly/1BEHajI`.

Algorithm	Cost per sentence	Accuracy
Soylent	\$1.38 (0.01)	84.41% (0.44)
BudgetFix	\$0.71 (0.01)	85.11% (0.42)
Budgeteer	\$0.75 (0.04)	85.28% (0.48)

Table 2: The average cost and accuracy (with 95% confidence value) of Soylent, BudgetFix (with optimally tuned parameters), and Budgeteer.

Complexity:	Low	Medium	High	V. High
Solved in Expl. Phase	39%	45%	27%	28%

Table 3: The percentages of sentences solved in the Exploration Phase of Budgeteer per sentence complexity.

As can be seen from Table 2, all three algorithms perform similarly — reaching approximately 85% correction accuracy. However, BudgetFix (when its parameters are optimally tuned) and Budgeteer, both reach this accuracy with only half of the cost. This result highlights the success of both of these algorithms in performing comparably to the state-of-the-art baseline, while significantly reducing the budget needed to achieve this performance.

Shortcomings of BudgetFix: Note, however, that BudgetFix is *manually tuned* to achieve comparable performance to Budgeteer. Crucially, BudgetFix requires three parameters to be tuned: $K_{\max}$, $L_{\max}$, and ε (while $K_{\max}$ and $L_{\max}$ in BudgetFix have a similar purpose in Budgeteer, ε is used to control the accuracy of estimation in the Find phase). As Figure 1 shows, the performance of BudgetFix can vary significantly if these parameters are poorly set. Moreover, note that the BudgetFix performance in Table 2 is based on its (manually-set) optimal parameter settings (i.e., $K_{\max}$ = $L_{\max}$ = 2 and ε = 0.1). However, BudgetFix's performance can drop to as low as 73% if these parameters are not set optimally (e.g., when $L_{\max} = 2$ and $K_{\max}$ and ε were changed only slightly to 3 and 0.2 respectively). This result highlights BudgetFix's sensitivity to setting its parameters, something that is overcome in Budgeteer. In contrast, Budgeteer automatically sets its parameters to fit each sentence according to its difficulty level, and thus, it does not suffer from this.

Advantages of Budgeteer: Table 3 is useful for understanding why Budgeteer is successful. Budgeteer's algorithm leverages its exploration phase to identify which sentences are relatively hard to correct and, therefore, allocate more micro-tasks to these sentences and fewer to easy ones. Recall that we consider the first stage to have found an "easy" sentence if all three people agree in the Find stage. Note that the low and medium categories have nearly half of the sentences being "solved" as a result of this condition (i.e., 39% and 45%). As is somewhat to be expected, sentences with lower structure complexity (see Table 1) were found easier

Difficulty:	Low			Medium			High			Very High		
Algorithm	Find	Fix	Verify	Find	Fix	Verify	Find	Fix	Verify	Find	Fix	Verify
Soylent	10.0	5.3	7.9	10	5.8	7.0	10	6.6	7.4	10	6.9	6.4
BudgetFix	5.0	4.0	8.0	5.0	4.2	8.0	5.0	4.2	8.0	5.0	4.4	8.0
Budgeteer	5.6	3.7	9.4	5.8	5.0	9.2	5.9	6.5	8.8	6.0	8.0	8.6

Table 4: The average numbers of micro-tasks (Find, Fix, and Verify) allocated by Soylent, BudgetFix (optimal), and Budgeteer.

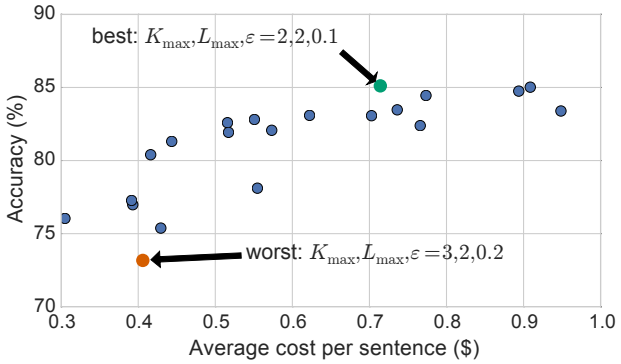

Figure 1: BudgetFix Performance at $B = \$1.50$

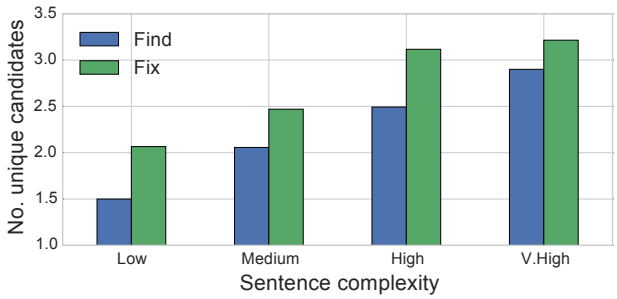

Figure 2: The average number of unique fix and find candidates identified by Budgeteer per sentence complexity.

to correct. However, this is not always the case, as the Low category had fewer sentences being solved in the exploration phase (39% versus 45% respectively). Moreover, more than 25% of the High and Very High sentences were also solved during this phase. This result shows that task complexity can, and should, be determined by the workers' performance during a deployment and not by external measures, including the sentence complexity. By so doing, Budgeteer can better identify which sentences are in fact in need of additional resources in the upcoming Fix and Verify micro-tasks. Among the sentences that were not solved in the first stage, in the second stage, however, we found that the number of unique Find and Fix identified by Budgeteer strongly correlated with the sentence complexity (see Figure 2).

To further illustrate the differences between Budgeteer, Soylent, and BudgetFix, Table 4 presents a full comparison of the numbers of Find, Fixes, and Verifies in all sentence categories. Given that both Soylent and BudgetFix do not consider budget allocations *across* workflows and consider each sentence independently, the number of micro-tasks in both of these algorithms only slightly changes across different sentence categories. In contrast, as Budgeteer uses

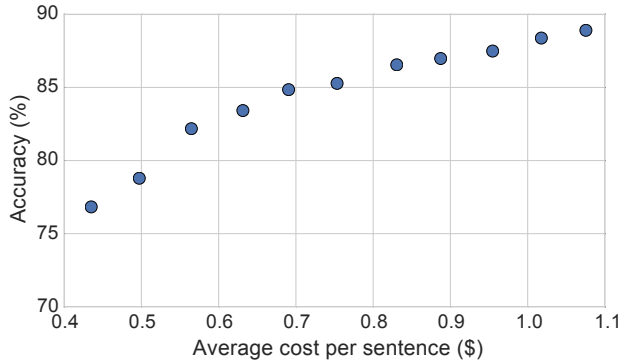

Figure 3: Budgeteer's accuracy vs cost per sentence.

a joint budget across all workflows, it dynamically redistributes its budget to address the sentences which its first stage found to be harder to correct. As per Table 3, more sentences were removed in Budgeteer's first stage in the low and medium categories, and therefore, it is not surprising that relatively large differences in the number of Fixes exist across different categories (ranging from 3.7 for the low category to 8.0 for the very high one). This result implies that a joint budget is better at adjusting to task difficulty.

Another key advantage of Budgeteer over BudgetFix is that only one parameter needs to be set, the total budget B^0. As Figure 3 demonstrates, when B^0 is increased, Budgeteers overall performance improves, albeit the improvement tails off in the long run. This is because the algorithm focuses progressively more of its budget on the hardest sentences. However, as several sentences in this dataset were not correctly solved by any of the workers, it was not possible to achieve 100% accuracy.

Conclusions

This paper presents Budgeteer, the first budget and task allocation algorithm that can efficiently deal with crowdsourced task allocation within multiple complex workflows under a budget constraint. We tested our algorithm within a canonical text error correction system, using the FFV workflow to formalise the inter-dependency between the micro-tasks. Here, the goal is to perform text correction on multiple sentences, each is represented by a FFV workflow. We benchmarked Budgeteer and show that it outperforms the benchmarks both in terms of cost and accuracy. Future work will look at extending Budgeteer to consider more complex workflows that may involve, for example, recursive tasks, or micro-tasks with varying costs.

Acknowledgement

We gratefully acknowledge funding from the UK Research Council for project 'ORCHID', grant EP/I011587/1.

References

AMT. 2010. Amazon mechanical turk. *http://www.mturk.com/*.

Audibert, J.-Y.; Bubeck, S.; and Munos, R. 2010. Best arm identification in multi-armed bandits. *COLT* 41–53.

Azaria, A.; Aumann, Y.; and Kraus, S. 2014. Automated agents for reward determination for human work in crowdsourcing applications. *Journal of Autonomous Agents and Multi-Agent Systems* 28:934–955.

Bernstein, M. S.; Little, G.; Miller, R. C.; Hartmann, B.; Ackerman, M. S.; Karger, D. R.; Crowell, D.; and Panovich, K. 2010. Soylent: a word processor with a crowd inside. In *UIST*, 313–322.

Boyd, S., and Vandenberghe, L. 2004. *Convex Optimization*. New York, NY, USA: Cambridge University Press.

Dai, P.; Lin, C. H.; Mausam; and Weld, D. S. 2013. POMDP-based control of workflows for crowdsourcing. *Artificial Intelligence (AIJ)* 202:52–85.

Flesch, R. 1948. A new readability yardstick. *Journal of Applied Psychology* 32(3):221–233.

Ho, C.-J., and Wortman-Vaughan, J. 2012. Online task assignment in crowdsourcing markets. In *AAAI'12*, 45–51.

Kalyanakrishnan, S.; Tewari, A.; Auer, P.; and Stone, P. 2012. PAC subset selection in stochastic multi-armed bandits. In Langford, J., and Pineau, J., eds., *ICML*, 655–662. New York, NY, USA: Omnipress.

Kamar, E.; Hacker, S.; and Horvitz, E. 2012. Combining human and machine intelligence in large–scale crowdsourcing. *AAMAS* 467–474.

Karger, D. R.; Oh, S.; and Shah, D. 2011. Budget-optimal task allocation for reliable crowdsourcing systems. *CoRR* abs/1110.3564.

Kaufmann, E., and Kalyanakrishnan, S. 2013. Information complexity in bandit subset selection. In *COLT*, volume 30 of *JMLR Workshop and Conference Proceedings*, 228–251. JMLR.

Kincaid, J. P.; Fishburne, R. P.; Rogers, R. L.; and Chissom, B. S. 1975. Derivation of New Readability Formulas (Automated Readability Index, Fog Count and Flesch Reading Ease Formula) for Navy Enlisted Personnel. Technical report.

Lasecki, W. S.; Bigham, J. P.; Allen, J. F.; and Ferguson, G. 2012. Real-time collaborative planning with the crowd. In *AAAI*.

Lin, C. H.; Mausam; and Weld, D. S. 2012. Dynamically switching between synergistic workflows for crowdsourcing. *AAAI'12* 87–93.

Little, G.; Chilton, L. B.; Goldman, M.; and Miller, R. C. 2009. Turkit: tools for iterative tasks on mechanical turk. *KDD-HCOMP* 29–30.

Ramchurn, S. D.; Huynh, T. D.; Venanzi, M.; and Shi, B. 2013. Collabmap: Crowdsourcing maps for emergency planning. In *5th ACM Web Science Conference (WebSci '13)*.

Simpson, E.; Roberts, S.; Psorakis, I.; Smith, A.; and Lintott, C. 2011. Bayesian combination of multiple imperfect classifiers. *NIPS*.

Tran-Thanh, L.; Stein, S.; Rogers, A.; and Jennings, N. R. 2012. Efficient crowdsourcing of unknown experts using multi-armed bandits. In *ECAI'12*, 768–773.

Tran-Thanh, L.; Venanzi, M.; Rogers, A.; and Jennings, N. R. 2013. Efficient budget allocation with accuracy guarantees for crowdsourcing classification tasks. *AAMAS'13* 901–908.

Tran-Thanh, L.; Huynh, T. D.; Rosenfeld, A.; Ramchurn, S.; and Jennings, N. R. 2014. Budgetfix: budget limited crowdsourcing for interdependent task allocation with quality guarantees. In *AAMAS'14*, 477–484.

Zhang, H.; Law, E.; Gajos, K.; Miller, R.; Parkes, D.; and Horvitz, E. 2012. Human computation tasks with global constraints. *ACM CHI*.